Blackbox and Scenario-Based Testing of Online Games Using Game Description Language

Chang-Sik Cho, Dong-Chun Lee, Kang-Min Sohn, Chang-Joon Park, and Ji-Hoon Kang

In this letter, we propose blackbox and scenario-based testing of multiplayer online games as well as simple load testing. Game testing is done from outside the source code, and the access to the source code is not required to testers because the game logic is described with a game description language and virtual game map. Instead of using a subset of the main game client for the test client, only game packet protocols and the sequence of packets are analyzed for new game testing. In addition, complex and various scenarios can be tested through combining defined actions. Scenario-based testing helps testers mimic real testing environments instead of doing simple load testing and improves the productivity of game testing.

Keywords: Multiplayer online game testing, load test, massive virtual users, blackbox testing, scenario-based testing, game description language, virtual game map.

I. Introduction

A massively multiplayer online game (MMOG) is a multiplayer video game that can be played via a game server over the Internet with players from around the world. In an MMOG, large numbers of players enter a single virtual world concurrently and interact with each other [1]. World of Warcraft [2] is the most representative of MMOGs.

MMOGs are deployed using a client-server system architecture and can actually be run on multiple servers. The stability and performance of game servers have become major issues in online games because the servers must be able to handle and verify a large number of connections. To meet such requirements for game servers, many testers are involved in game testing, and a large number of game players actually participate in beta testing before a game is deployed. In order to reduce such testing time by emulating server loads, testing automation has been used [3], [4].

Traditional server load testing technologies, such as LoadRunner [5] and QALoad [6], have been applied to online game testing. They can emulate hundreds or thousands of concurrent users by reusing captured packets, and this helps reduce the costs and time required to test and deploy new applications. However, various game actions cannot be easily simulated by simply and partly modifying the game packets of an actual player because an online game server employs complicated logic compared with a general server application [7]-[9].

Thus, The Sims Online (TSO) automation [10] and the Virtual Environment Network User Simulator (VENUS) system [11] used a virtual client engine with a subset of actual game clients. The TSO team used a subset of the main game client to create a test client in which the graphical user interfaces (GUIs) are mimicked via a script-driven control system. Also, the VENUS system used dummy game clients created using the VENUS Software Development Kit. Both TSO automation and the VENUS system have saved many person-hours, leading to better game quality, and have been more efficient than manual testing. However, they both have some weak points.

First, the VENUS system and TSO automation are whitebox approaches [3], which require access to the source code of the game client for the test client code. Although using an existing game client has minimized the need to create new codes, the test client code should be rewritten for new game testing. Game testers typically do not read the game code, and blackbox testing is the most cost-effective way to test the

Manuscript received May 18, 2010; revised Nov. 3, 2010; accepted Nov. 29, 2010.

Chang-Sik Cho (phone: +82 42 860 5942, email: cscho@etri.re.kr), Dong-Chun Lee (email: bluepine@etri.re.kr), Kang-Min Sohn (email: sogarian@etri.re.kr), and Chang-Joon Park (email: chjpark@etri.re.kr) are with the Contents Research Division, ETRI, Daejeon, Rep. of Korea.

Ji-Hoon Kang (email: jhkang@cnu.ac.kr) is with the Computer Engineering Department, Chungnam National University, Daejeon, Rep. of Korea.

doi:10.4218/etrij.11.0210.0172

extremely complex network of system [3]. Thus, the testers are required to test games without access to the game source code.

Second, the VENUS system and TSO automation do not support scenario-based testing. Their testing has been achieved using the unit of the simple command script. Although both have provided more flexible testing actions compared with traditional server load testing technology, it is difficult for them to test interactive actions such as multiuser collaborative play. Complex and varying test scenarios, such as party play and waypoint movement, are still not supported. However, game testers want to simulate complex scenarios and mimic real testing environments instead of doing simple load testing.

In this letter, along with simple load testing, we propose blackbox and scenario-based testing of online games. As in existing approaches, massive virtual clients automatically generate packet loads to test the stability of game servers. However, by describing game logics using the game description language and virtual game map, the test client code itself does not need to be rewritten for new game testing. Also, complex scenarios, such as iterative attack, party play, and waypoint movement, can be tested by combining actions.

II. Scenario and Game Description Language

In our approach, testing is automated by using scenarios. To generate a heavy load and provide a set of stresses to the game servers, a plurality of virtual users should be created and controlled. A large number of virtual users can be generated with group and action commands being applied to the group of virtual users at the same time. An example of game testing scenarios is shown in Fig 1. A script language with simple control constructs and time constraint constructs is used for describing test scenarios. So, the testers can minimize the simple and repetitive work by using script.

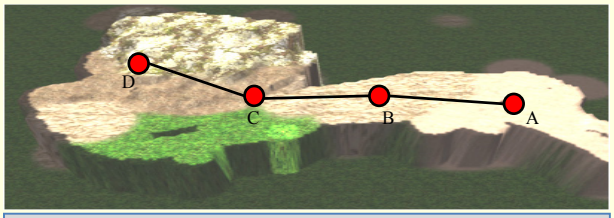
From the viewpoint of an individual, a test scenario is a sequential list of actions with corresponding positions. These actions are meaningful user activities, such as logins, movement, attack, trading, and so on. Figure 2 shows a conceptual view of a test scenario from the viewpoint of an individual user. As shown in the example, a test scenario is made up of a sequence list of position and action pairs.

In our method, the position information is represented in a virtual game map and the actions are represented using the game description language. That is, all of the game logics are described by defining the game description language and virtual game map, and thus the test client code does not need to be rewritten when a new game is to be tested.

A virtual game map is a collection of x, y, and z positions and contains texture information and building objects. The process by which a virtual game map is generated is as follows. At first,

```
Per 1 second, login 10 users continuously until the number of the users is reached to N_MAX.
For 1 hour, individual users repeatedly do the hunting and looting, gathering, and trading.
For 1 hour, per 1 second, repeatedly logout 10 users and login new 10 users.
```

Fig. 1. Example of game testing scenarios.




```
Go to position A, B, C and D sequentially.
In A, generate user and enter game world.
In B, hunt 70 mobs and loot the item.
In C, gather the 30 minerals, then go to D position.
There, sell all your gathered items.
```

Fig. 2. Test scenario with positions and actions.

```
Game_Description_Language := (Protocol, Action, Generation_Rules)
        Protocol := (Element)
        Action := (Time_Constraint, Socket, Protocol, Parameter)
        Generation_Rules := (Endianess_Rule,  Encryption_Rule,
                             TimeStamp_Rule, HeartBeat_Rule, ...)
            Element := (Name, Type, Length, Byte_string)
```

Fig. 3. Syntax of game description language.

the virtual game map is blank. The actual game players or virtual users move randomly and en masse around the game world. The game world is gradually searched using the position information gathered by game users' movements. The packet analyzing tool determines which areas the game users are and are not allowed to visit. Finally, the tester can modify the map textures and place building objects such as villages, stores, and hunting grounds manually.

The game description language consists of actions, packet protocols, and packet generation rules. The actions consist of a series of packet protocols, and the packet protocols define the individual packet messages transmitted between the game server and game client. The packet generation rules define the overall packet structure rules such as encryption, endianness, timestamp, and so on. Figure 3 shows the syntax of the game description language.

III. Experimental Results

1. Implementation Results

Our system consists of a packet analyzing tool and a virtual

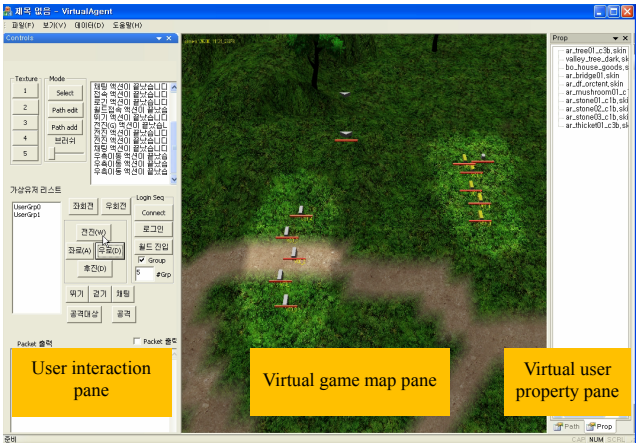


Fig. 4. Virtual user control tool.

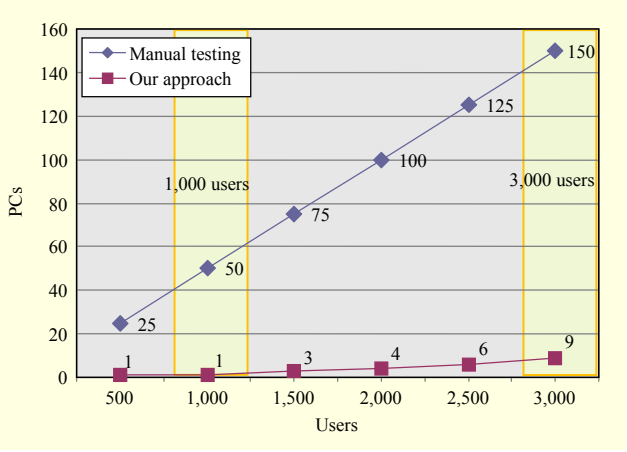


Fig. 5. Number of required PCs.

user control tool. Actual game packets transmitted between a game client and game server are captured and stored in the packet analyzing tool. Then, the stored packet lists are analyzed, and the game description language is generated manually. To provide an efficient analysis of a game protocol, a packet analyzing tool provides various functionalities, such as coloring, filtering, type conversion utilities, and packet difference checking utilities.

Figure 4 is a screenshot of the virtual user control tool. A virtual user is an entity of a single online game user. To the main game server, these test clients look identical to an actual connected game player. In the virtual user control tool, there are three window panes: a user interaction pane, virtual game map pane, and virtual user property pane. The virtual user control tool generates an actual load when the tester inputs actions or scenarios in the user interaction pane. In the virtual game map in the center, virtual users and mobs are displayed according to their position using circles and triangles. The red bar below the shape indicates the health points or life bar of the game objects, and the color of the shape indicates the states of the objects, such as in-combat, walking, dead, and so on. When the tester selects a virtual user in the virtual game map, the detailed properties of the virtual user are shown in the property pane window.

2. Experimental Results

To verify the effectiveness of our method, our system has been applied to several online games. The applied game genres [1] include the massively multiplayer online role-playing game (MMORPG), multiplayer online game (MOG), and casual game. For MOGs and casual games, the testing is rather simple; it includes entering, leaving, and re-entering rooms repetitively, and a virtual game map is not used.

In this subsection, we explain the experimental results applied to ELMA, which is a typical MMORPG in its alpha testing period. The login/logout, movement, attack, looting, gathering, and trading actions are successfully defined using the game description language in the packet analyzing tool. The virtual user control tool generated and controlled 1,000 ELMA virtual users in a 2.8 GHz Intel Core2 Quad CPU PC with a 100 Mbps Ethernet card.

Figure 5 illustrates the changes in the number of PCs required when massive game clients enter the game world. The number of virtual clients is increased from 500 up to 3,000. A single PC can handle 20 ELMA clients, so the number of required PCs for manual testing can be easily calculated. Our system requires only 9 PCs for testing 3,000 users, while manual testing by human testers requires 150 PCs. In our system, the number of required PCs is not linear but quadratic because the movement and notification messages are in proportion to the number of users. The bottleneck of our system is the amount of network traffic, and a PC can handle 50 Mbps game packets in ELMA game.

It took only 24.8 min (1,485 s) in our system when a scenario similar to Fig. 2 was tested for 1,000 users, while it took 6.7 hours (24,000 s) for manual testing. It took 24 s for a single actual gamer to perform the scenario, and 32 windows keyboard/mouse interactions were required. Our system is 16 times faster than manual testing. Also, we found there is an average of 3.5 socket connection errors and 3% authentication errors when the number of users reaches 2,000.

The result shows that bugs can be found efficiently in massive load testing, and many person-hours can be saved by replacing time intensive work with our system. Of course, there is always the human factor that can never be replaced with automated testing [12]. So the testers should adequately use both human testing and tool testing, rather than relying wholly on our system.

Table 1 shows a comparison with previous approaches. Our

Table 1. Comparison with previous approaches.

Categories \ Approaches		HP LoadRunner [5]	TSO automation [10]	VENUS [11]	Our approach
Blackbox testing	Main idea	Reuse captured data	Reuse game client code	Reuse game client code	Capture, analyze, and generate GDL
	Backbox/whitebox	Blackbox	Whitebox	Whitebox	Blackbox
	Requiring source code	No	Yes	Yes	No
	Reusable to another game	Yes (but very hard)	No	No	Yes
Scenario-based testing	Game map support	No	No	No	Yes
	Unit of testing	Protocol	Action	Action	Scenario
	Flexibility to various test scenario	Hard	Easy	Moderate	Easy

approach is compared with existing game testing approaches from the viewpoint of blackbox and scenario-based testing. TSO automation and the VENUS system are based on whitebox testing. Therefore, their approaches can be applied to game developers only and are not adequate for game testers because the developed game client code must be provided and rewritten for game testing. Though LoadRunner from Hewlett-Packard supports blackbox testing by reusing captured packet data, it is difficult for it to be applied to game testing because an online game server employs complicated logic compared with a general server application, such as a web server.

IV. Conclusion

In this letter, along with simple load testing, we propose blackbox and scenario-based testing of online games. In previous test automation, developers were required to provide their client code to the game testers because a test client code is based on a dummy game client code without a GUI. In our approach, the game logic is described using game description language and a virtual game map, so the test client code does not need to be rewritten when a new game is to be tested. The virtual clients automatically generate a packet load according to the game description language. Moreover, complex scenarios, such as iterative attack and looting, party play, and waypoint movement, can be tested by combining actions. In our system, the testing unit is scenario-based instead of command script and we use the game map for intuitive user interface.

To verify the effectiveness of our method, we have applied our system to several online games. When applied to the online game ELMA, various actions, such as the login, attack, and trading, are defined using the game description language in the packet analyzing tool, and the virtual user control tool generated massive virtual users and gave some clues for server error. Experimental results show that a remarkable amount of time can be saved by replacing time intensive work with our system.

Future research is planned to improve our system, provide more intelligent test scenarios, and create a standard model for online game testing.

References

[1] Game genres. http://en.wikipedia.org/wiki/Game_genres

[2] Blizzard - World of Warcraft. http:// www.worldofwarcraft.com

[3] C.P. Schultz, R. Bryant, and T. Langdell, *Game Testing All In One*, Thomson Course Technology PTR, 2005.

[4] L. Mellon et al., "Large-Scale Engineering for Online and Offline Games," *GDC*, Spring 2007.

[5] HP LoadRunner software - HP - BTO Software. http://www.hp.com

[6] QALoad. http://www.compuware.com/

[7] S. Elbaum, S. Karre, and G. Rothermel, "Improving Web Application Testing with User Session Data," *Proc. 25th Int. Conf. Software Eng*, May 3-10, 2003, pp. 49-59.

[8] Y.-T. Han and H.-S. Park, "Game Traffic Classification Using Statistical Characteristics at the Transport Layer," *ETRI J.*, vol. 32, no. 1, Feb. 2010, pp. 22-32.

[9] K. Shin et al., "Transformation Approach to Model Online Gaming Traffic," *ETRI J.*, vol. 33, no. 2, Apr. 2011, pp. 219-229.

[10] L. Mellon, "Automated Testing of Massively Multi-player Systems: Lessons Learned from The Sims Online," *GDC*, Spring 2003.

[11] B.H. Lim, J.R. Kim, and K.H. Shim, "A Load Testing Architecture for Network Virtual Environment," *Proc. 8th Int. Conf. Adv. Commun. Tech.*, Feb. 20-22, 2006, pp. 848-852.

[12] E. Rees and L. Fryer, *Best Practices in Quality Assurance/Testing*, IGDA Business Committee, Apr. 2003.